Design Rationale

Object-Oriented Programming Principle implementations:

New Class Implementations:

S O U L S

Soul class inherits from the Item class as Items represent an object in the Gameworld. As it is picked up and dropped which can be done through utilising getPickUpAction() and getDropAction() method in Items. When Souls are dropped by dead Enemies, it is instantiated, allowing the player to perform the method getPickUpAction(). This is done to prevent repetitive code. Hence, the Don't Repeat Yourself (DRY) principle, where a piece of logic will have its unique and specific functions within a system. There will also be a TokenOfSouls attribute that stores the Souls for when the player dies, meaning the number of souls in the inventory of the Player before they die is stored in this attribute.

E N E M Y

Enemy class is an abstract class extending the Actor class, representing the children classes and acts as a base code for its child classes: Undead, Skeletons and Lord of Cinder. To ensure that an instance of the Enemy class is not instantiated. The death of Enemies will drop Souls, the currency of the game. Each Enemy will have the base Soul count of 50 for Undead, 250 for Skeletons and 5000 for Lord of Cinder. These values are stored as a constructor parameter for creating instances of each enemy. The main objective is for the Player to pick up the Souls that are dropped by Enemies after they are killed and added into the Player's inventory and most importantly, the attacks between the Player and Enemy. The damages dealt by the Enemy can be stored in this class to prevent duplicate code in the Enemy children classes, instead, the children classes can just override this method. This is again an example of the implementation of the DRY principle as well as avoiding the excessive use of literals as it is not hard-coded, making the maintenance of the code smoother and easier in the future.

Enemy class is created to follow the Reduce Dependency Principle as the Actor class is initially inherited by the Player, Undead, Skeleton, LordOfCinder and Ground class. Hence, the Enemy class will inherit the Actor class instead, which is then inherited by its children classes like the Undead and EnemyWithWeapon (inherited by Skeletons and Yhrom) reducing dependency on the Actor class.

Enemy class will also be inclusive of an Arraylist of the Actor data type, keeping track of if the Actor is a Player or an Enemy is close together to determine the behaviour. As this attribute will be looped through to determine if Enemy will use AttackAction() if the Player is in target range. Thus, also reducing the use of duplicate code and utilising the DRY principle.

S K E L E T O N

Skeleton class will extend the Enemy class as it deals damage and displays phrases like "thwacks" or "punches". Each skeleton will also deal base damage to the player signifying how it will override the method inherited from the Enemy class. This class will implement a spawnEnemies() method which spawns 4 to 12 skeletons on the map. It is given the range of Skeletons that need to be spawned on the map and utilises WanderBahvaiour() to place the Skeleton in a random location of the map.

E N E M Y W I T H W E A P O N

EnemyWithWeapon class is implemented and inherits the Enemy class and implements the Undead class does not wield weapons unlike Lord of Cinder and Skeletons. This implementation will follow the Liskov Substitution Principle (LSP) as the constructor in the Enemy Class, will have a parameter asking for weapons that the Enemy is possibly wielding. This demonstrates how the superclass object is replaceable with the subclass object without affecting the functionality of Enemy class. Therefore, the Enemy will still be able to attack the Player.

B O N F I R E

Bonfire is an extension of Ground. Overriding the canActorEnter() method and setting the return boolean value to False if the Actor is an Enemy. - Also resets estus flask, enemy positions and player health.

V E N D O R

This class will hold the methods in order for the Player to trade Souls for upgrades or Weapons. Vendor implements Souls as Souls is the object that interacts between Player and Vendor for the upgrades or weapon purchases. As the Player purchases a Weapon, the SwapWeaponAction which is a part of PickUpItemAction() is then called. This occurs as the Player can only wield one weapon at the time.

C E M E T E R Y

Cemetry class extends Ground class as the class represents

New Method Implementations:

TargetActorInRange()

With the return type of boolean. This method is implemented to check if the Player is within range of the Enemy, when boolean returns True, the Enemy will use AttackAction() to attack the player, decreasing hitPoints. Target Actor refers to Player and this method is implemented in the Enemy class as all types of Enemy will check if the Player is within it's range.

SpawnEnemies()

As mentioned above, this method is responsible for spawning the targeted number of enemies in the GameMap. Nested under the Enemy class, it is given the number range of Enemy type that needs to be spawned.

HasRevived()

This boolean return type method is implemented int he Skeleton class. As Skeletons has a 50% success rate of healing itself. When the method returns False, meaning the Skeleton has not revived itself, it will heal to it's max_heal_points. Else, when the method returns True, the Skeleton is thenm removed form the GameMap.

<u>New classes and methods:</u>

- Skeletons
- Souls inherit from items
- Token of Souls
- Vendor

- Bonfire
- Cemetery
- Enemy
- CindersOfLord
- TargetActorInRange() → compares the distance between the actors and whether they are in range with each other → returning true if they are
- spawnEnemies() in Enemy class → spawns the number of enemies required in the whole map randomly.
- hasRevived - seeing if skeleton revives.
- hasBurn() - below 50% hit points, he can set the dirt ground on fire.